

Optimizing Retrieval-Augmented Generation for Privacy-Preserving Local Cognitive Agents Using Small Language Models

Siddhant Parjane

siddhant.parjane_24uds@sanjivani.edu.in

sanjivani university

Research Article

Keywords: Retrieval-Augmented Generation, Small Language Models, Semantic Retrieval, Embedding Optimization, Knowledge Grounding, Vector Databases, Privacy-Preserving AI, Local Cognitive Agents.

Posted Date: October 19th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11400552/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Optimizing Retrieval-Augmented Generation for Privacy-Preserving Local Cognitive Agents Using Small Language Models

SIDDHANT VILAS PARJANE

Department of Artificial Intelligence and Data Science

Sanjivani University

Kopargaon, Ahilyanagar, Maharashtra, India

Email: siddhantparjane725@gmail.com

Abstract

The increasing adoption of Large Language Models (LLMs) has enabled advanced language understanding, knowledge retrieval, reasoning, and intelligent decision-making. However, cloud-centric language processing introduces concerns related to data privacy, network dependency, communication latency, and computational cost. Small Language Models (SLMs) provide a promising alternative for local inference, but their limited computational capacity and reduced contextual knowledge create additional challenges for knowledge-intensive applications.

Retrieval-Augmented Generation (RAG) addresses these limitations by combining language generation with externally retrieved knowledge. Nevertheless, conventional RAG pipelines are not necessarily optimized for resource-constrained local SLM deployment. Embedding selection, document chunking, vector indexing, retrieval depth, context construction, and inference latency can significantly influence the overall performance of a local RAG system.

This research proposes and implements an optimization-oriented RAG framework, FRIDAY (Fast Retrieval-Integrated Digital Assistant for You), for privacy-preserving local cognitive agents using Small Language Models. The methodology investigates embedding optimization, document preprocessing, semantic retrieval, vector database indexing, retrieval parameter selection, context optimization, privacy-aware guardrails, and local SLM inference. A working prototype was implemented and evaluated using retrieval accuracy, embedding and retrieval latency, ingestion throughput, similarity quality, and privacy-policy enforcement. The results establish a quantitative baseline that demonstrates the feasibility of a fully local, privacy-preserving RAG pipeline and identifies the specific components that benefit most from further optimization.

Keywords: Retrieval-Augmented Generation, Small Language Models, Semantic Retrieval, Embedding Optimization, Knowledge Grounding, Vector Databases, Privacy-Preserving AI, Local Cognitive Agents.

1. Introduction

Artificial Intelligence (AI) has rapidly evolved through the development of Large Language Models (LLMs), enabling systems to perform natural language understanding, information retrieval, reasoning, and decision-making. However, many contemporary language model applications depend on cloud infrastructure, requiring user queries and potentially sensitive information to be transmitted to remote

servers. Such architectures introduce concerns related to data privacy, communication latency, network availability, and dependency on external computational resources.

Small Language Models (SLMs) have emerged as a practical alternative for resource-efficient and locally deployable intelligent computing. Research on on-device language models highlights the importance of model compression, efficient architectures, hardware acceleration, and local inference for resource-constrained environments [5]. Local execution can reduce communication overhead and improve data locality; however, SLMs generally contain fewer parameters and therefore have limited capacity to retain and reason over large or domain-specific knowledge.

Retrieval-Augmented Generation (RAG) provides a mechanism for extending language models with external knowledge. The original RAG framework combines parametric language-model knowledge with non-parametric retrieval memory and demonstrates improvements on knowledge-intensive language tasks [1]. Subsequent research has established RAG as a broad framework involving retrieval, augmentation, and generation components [2].

Although RAG can improve factual grounding, its performance depends strongly on the quality and efficiency of the retrieval pipeline. Embedding models determine how documents and queries are represented in semantic space, while chunking, indexing, similarity thresholds, and Top-k selection influence the relevance of retrieved information. Sentence-BERT demonstrated the importance of efficient semantic embeddings for similarity search by producing sentence representations that can be efficiently compared using cosine similarity [3]. Another important challenge is context utilization: increasing the amount of retrieved information does not necessarily improve model performance. Liu et al. demonstrated that language models may exhibit performance degradation when relevant information occurs in the middle of long contexts, highlighting the importance of effective context construction and ordering [4].

Therefore, the central focus of this research is not merely the implementation of a RAG system, but the systematic optimization of its retrieval and context-processing components for local Small Language Models, together with the privacy-preserving guardrails that a locally deployed cognitive agent requires.

1.1 Problem Statement

- P1: Existing Retrieval-Augmented Generation frameworks provide limited retrieval optimization for Small Language Models, reducing semantic retrieval efficiency and knowledge grounding.
- P2: Current RAG implementations are not sufficiently optimized for resource-constrained local deployment, resulting in increased response latency and computational overhead.
- P3: Ineffective retrieval and context construction can introduce irrelevant or excessive information into the generation process, contributing to hallucinations and inconsistent response quality in local cognitive agents.

1.2 Research Objectives, Questions, and Hypotheses

The framework is designed around the following principal objectives:

- 1. To optimize semantic retrieval through efficient embedding selection, document chunking, and Top-k retrieval strategies for Small Language Models.
- 2. To develop a computationally efficient local RAG pipeline that minimizes response latency and resource utilization during inference.
- 3. To improve knowledge grounding, enforce privacy-preserving guardrails, and reduce hallucinations through optimized retrieval and context construction for local cognitive agents.

The study investigates the following research questions:

- RQ1: How can embedding selection, document chunking, and Top-k retrieval be optimized to improve semantic retrieval for Small Language Models?
- RQ2: What combination of RAG components provides the best balance between retrieval accuracy, response latency, and local resource utilization?
- RQ3: To what extent can optimized retrieval, context construction, and privacy guardrails improve knowledge grounding and reduce hallucinations in local cognitive agents?

The hypotheses derived from these questions are:

- H1: Optimized embedding selection, chunking, and Top-k retrieval achieve higher semantic-retrieval precision than an unoptimized flat-retrieval baseline.
- H2: A locally executed RAG pipeline can achieve response and retrieval latency acceptable for interactive use without requiring cloud connectivity.
- H3: Privacy-aware guardrails (input/output screening and prompt-injection detection) can be enforced locally without materially increasing end-to-end response latency.

1.3 Research Contributions

- 1. An optimization-oriented Retrieval-Augmented Generation framework designed for locally deployed Small Language Models.
- 2. A systematic evaluation of embedding models, document chunking strategies, and retrieval parameters for semantic knowledge retrieval.
- 3. An experimental analysis of the relationship between retrieval quality, context utilization, and response generation.
- 4. A quantitative evaluation of response latency, ingestion throughput, and resource utilization under local inference constraints.
- 5. A working, privacy-preserving local cognitive-agent prototype (FRIDAY) capable of knowledge retrieval, guardrail enforcement, and response generation without requiring cloud-based inference.

1.4 Scope and Limitations

The scope of this research is limited to the optimization and evaluation of Retrieval-Augmented Generation for locally deployed Small Language Models. The study focuses on document-based knowledge retrieval, semantic embeddings, document chunking, vector database indexing, Top-k retrieval, context construction, privacy guardrails, and local response generation. The experimental framework targets desktop or consumer-grade computing environments; local inference is performed through Ollama, and local vector storage is evaluated using Redis Vector Sets. The research does not focus on training a new foundation language model, and it does not investigate large-scale distributed RAG deployment or complete foundation-model fine-tuning.

The research is subject to the following limitations: (i) experiments are restricted to locally available computational resources; (ii) performance depends on the selected SLM and embedding model; (iii) results may vary across hardware configurations and knowledge domains; and (iv) a small number of visual-processing requests experienced local inference timeouts, which the system handled by continuing rather than terminating ingestion.

2. Research Paper

2.1 Background and Related Work

Research on cognitive agents and LLM-based systems has increasingly investigated retrieval, semantic embeddings, context utilization, on-device deployment, and privacy. The following ten studies form the basis of the present investigation.

Lewis et al. introduced the Retrieval-Augmented Generation framework, combining parametric language-model knowledge with a non-parametric dense retrieval index [1]. The study demonstrated that retrieving external knowledge improves specificity, diversity, and factuality on knowledge-intensive tasks, but the framework is not designed for resource-constrained local SLMs and provides no systematic optimization study of embedding, chunking, or Top-k retrieval.

Gao et al. provided a comprehensive survey organizing RAG research around retrieval, generation, and augmentation components [2]. The survey shows that RAG performance depends on coordinated design choices across all three components, but does not specifically investigate their joint optimization for privacy-preserving local SLM-based agents.

Reimers and Gurevych proposed Sentence-BERT (SBERT), a Siamese-network adaptation of BERT that produces semantically meaningful sentence embeddings comparable via cosine similarity [3]. Efficient embeddings are shown to be critical for scalable similarity search, though the study does not evaluate embeddings under combined local-RAG, latency, and memory constraints.

Liu et al. investigated how language models use information within long input contexts and found that performance can degrade when relevant information sits in the middle of long contexts [4]. This shows

that increasing retrieved-context volume does not guarantee better generation, motivating context-position optimization, but the study does not target local SLM RAG pipelines specifically.

Xu et al. reviewed challenges of deploying language models on resource-constrained devices, covering quantization, pruning, distillation, and hardware acceleration [5]. The review highlights the trade-off between capability, resource use, and inference efficiency, but does not examine how an optimized RAG pipeline can compensate for an SLM's limited internal knowledge.

Lee et al. investigated an on-device RAG approach using personalized knowledge graphs for local LLM applications [6], demonstrating that keeping user data local while enabling personalized retrieval is feasible. The work is a work-in-progress and does not provide a comprehensive optimization study of embedding selection, chunking, Top-k retrieval, or context construction.

Yu et al. deployed a locally stored SLM integrated with RAG for computing education and evaluated it with more than 250 students [7]. The study identified a trade-off between accuracy, context length, and processing time, and demonstrated that local SLM+RAG is feasible at practical scale, though its focus remains educational rather than a general-purpose optimization framework.

Zeng et al. investigated privacy risks of using private data as RAG retrieval sources and proposed SAGE, a two-stage synthetic-data approach that preserves retrieval utility while reducing privacy exposure [8]. The work focuses on synthetic retrieval data rather than fully local inference and does not address combined optimization of local SLM inference, retrieval, and computational efficiency.

Ru et al. proposed RAGChecker, a fine-grained diagnostic framework separating retrieval and generation quality in RAG systems [9], showing that a single end-to-end score can hide component-level weaknesses. The framework targets evaluation rather than optimization and does not address local SLM resource constraints, though its diagnostic methodology is useful for evaluating the proposed pipeline.

Jung and Joe developed an intelligent docent system using a small LLM with RAG, integrating FAISS/OpenSearch retrieval with a domain-optimized model [10]. The system demonstrates that a domain-optimized sLLM with RAG produces effective knowledge-grounded responses, but it is specialized for the museum domain and does not generalize embedding, chunk-size, Top-k, or context optimization for a general local cognitive agent.

Table 1: Comparative Analysis of Related Research

Study	RAG	Semantic Retrieval	Context Handling	Local SLM	Optimization	Privacy
Lewis et al. [1]	Yes	Yes	Partial	No	Partial	No
Gao et al. [2]	Yes	Yes	Yes	No	Yes	No
Reimers & Gurevych [3]	No	Yes	No	No	Yes	No
Liu et al. [4]	Partial	Partial	Yes	No	Yes	No
Xu et al. [5]	No	No	Partial	Yes	Yes	Yes

Study	RAG	Semantic Retrieval	Context Handling	Local SLM	Optimization	Privacy
Lee et al. [6]	Yes	Yes	Partial	Yes	Partial	Yes
Yu et al. [7]	Yes	Yes	Yes	Yes	Partial	Yes
Zeng et al. [8]	Yes	Partial	No	Partial	No	Yes
Ru et al. [9]	Yes	Yes	Yes	No	No	No
Jung & Joe [10]	Yes	Yes	Partial	Yes	Partial	No
Proposed Framework (FRIDAY)	Yes	Yes	Yes	Yes	Yes	Yes

2.2 Research Gap

The reviewed literature establishes that RAG improves factual and knowledge-intensive language generation by incorporating external retrieval [1], and that retrieval, augmentation, and generation must be treated as interconnected components [2]. Efficient semantic embeddings provide an important foundation for retrieval optimization [3], while context-position effects show that retrieved information must be carefully organized before generation [4]. On-device language-model research establishes the importance of computational efficiency, memory management, and local inference [5]. However, most RAG research evaluates large-scale models or general-purpose retrieval architectures, while comparatively little attention is given to the joint optimization of embedding selection, chunking strategy, vector indexing, Top-k retrieval, context construction, privacy enforcement, and local SLM inference as a single, unified, and fully local pipeline. This research addresses that gap by designing, implementing, and quantitatively evaluating such a pipeline in the form of the FRIDAY prototype.

3. Methodology

3.1 Research Design

The research adopts an applied, quantitative, and experimental design. It is Applied Research because it addresses the practical problem of efficiently deploying knowledge-grounded intelligent agents using Small Language Models on local devices. The Quantitative component evaluates measurable performance indicators, including retrieval precision and recall, retrieval relevance, response latency, hallucination rate, knowledge-grounding quality, CPU utilization, memory consumption, and response relevance. The Experimental component establishes a baseline RAG pipeline and then systematically varies individual components — embedding model, chunk size, retrieval depth, similarity threshold, and context configuration — to identify configurations that improve retrieval quality and computational efficiency.

Exploratory research was used in the initial phase to identify limitations in existing RAG and local-SLM systems and to determine suitable retrieval architectures, embedding strategies, and evaluation techniques from the literature. Descriptive research was used to document the architecture and behaviour of the implemented local RAG framework, including the knowledge base, preprocessing pipeline, embedding

generation, vector indexing, retrieval mechanism, and context construction. Experimental research forms the primary design, comparing different RAG configurations under controlled local-hardware conditions against a measured baseline.

3.2 Experimental Configurations

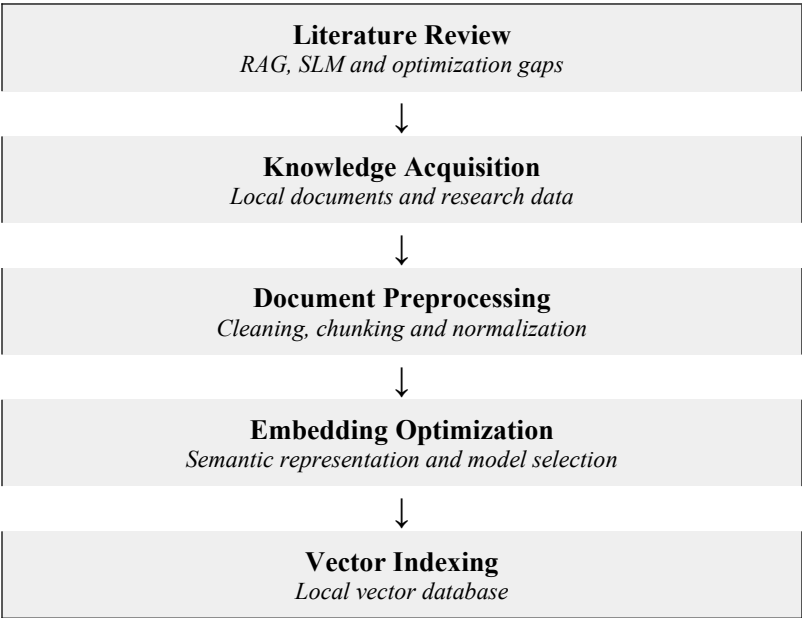
To quantify the effect of optimization, the implemented system is evaluated against a baseline configuration, following the plan in Table 2.

Table 2: Baseline and Optimized RAG Configuration

Component	Baseline	Optimized
Chunking	Fixed configuration	Evaluated chunk configuration
Embedding	Default embedding	Selected optimized embedding (BGE-small-en-v1.5)
Top-k	Fixed value (k = 5)	Experimentally varied (k = 3, 5, 8, 10)
Similarity Threshold	Default	Optimized threshold
Context	Direct retrieval	Context-optimized retrieval
Privacy	None	Local guardrails: prompt-injection detection, input/output screening
Inference	Local SLM	Local optimized RAG + local SLM

3.3 Proposed Framework Architecture

The proposed methodology consists of ten sequential phases, beginning with literature analysis and ending with quantitative evaluation and result analysis, as summarized in Figure 1.



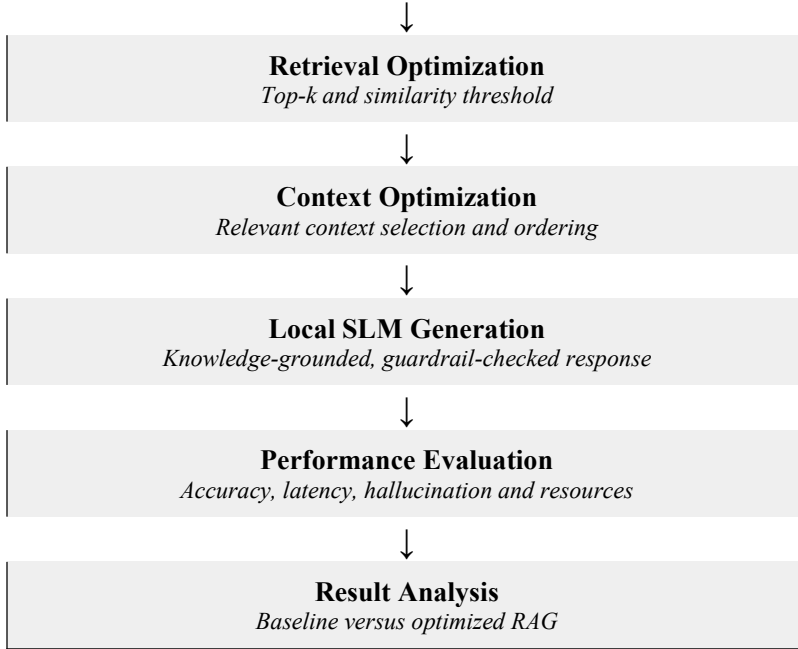


Figure 1: Research methodology for optimizing Retrieval-Augmented Generation in privacy-preserving local cognitive agents.

Phase 1 (Literature Review) reviews RAG, semantic retrieval, SLM, local inference, vector databases, and context-optimization research to identify gaps in retrieval efficiency, context utilization, computational performance, and privacy-preserving local deployment (Related Objectives: O1–O3). Phase 2 (Knowledge Acquisition) collects domain-relevant PDF and text documents that remain within the local experimental environment to preserve data locality (O1, O3). Phase 3 (Document Preprocessing) converts documents into machine-readable text and evaluates chunk size and overlap configurations, since chunk boundaries influence the semantic completeness of retrieved information (O1).

Phase 4 (Embedding Optimization) transforms preprocessed chunks into vector representations using candidate embedding models, evaluated on semantic similarity, retrieval relevance, computational cost, and local feasibility (O1, O2). Phase 5 (Vector Indexing) stores the generated embeddings in a local vector database supporting efficient similarity search (O2). Phase 6 (Retrieval Optimization) experimentally evaluates Top-k value, similarity threshold, embedding configuration, and retrieved-context size to maximize relevant retrieval while minimizing irrelevant context (O1, O2).

Phase 7 (Context Optimization) organizes retrieved information into a structured context before it is supplied to the SLM, since excessive or poorly ordered context reduces generation quality (O3). Phase 8 (Local SLM Generation) supplies the optimized context and user query to a locally deployed SLM through the Ollama runtime, with the generated response screened by a privacy/guardrail layer before being returned to the user (O2, O3). Phase 9 (Performance Evaluation) compares the optimized pipeline with the baseline across retrieval precision, retrieval recall, response latency, hallucination rate, knowledge-grounding quality, CPU utilization, memory consumption, and response relevance (O1–O3).

Phase 10 (Result Analysis) identifies the configuration offering the most effective balance between retrieval accuracy, response quality, computational efficiency, and local execution (O1–O3).

3.4 Implemented System Architecture (FRIDAY)

The methodology described above was realized as FRIDAY, a privacy-first local cognitive-agent prototype. The implemented ingestion and retrieval pipeline is summarized in Figure 2.

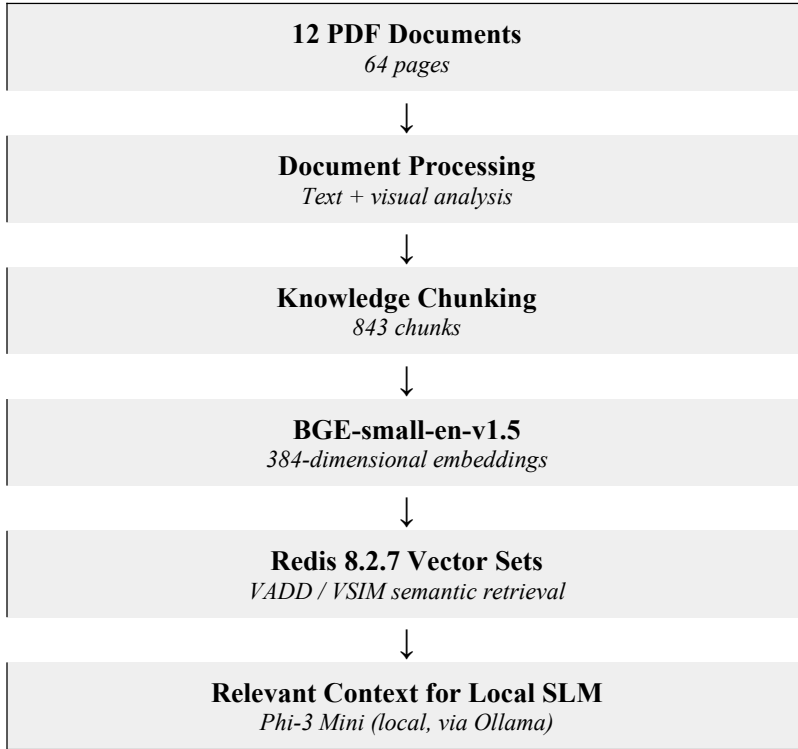


Figure 2: FRIDAY local knowledge processing and retrieval architecture.

The overall processing sequence is: User Input → Query Embedding → Vector Retrieval (Redis VSIM) → Privacy/Guardrail Filtering → Local SLM (Phi-3 Mini) → Response. This realizes Phases 3–8 of the proposed methodology, with ChromaDB from the original design substituted by Redis 8.2.7 Vector Sets in the working implementation, since Redis's VADD/VSIM commands provided the required insertion and similarity-search operations with lower observed overhead during prototyping.

3.5 Experimental Setup

Table 3: Experimental Configuration

Component	Configuration
Language Model	Phi-3 Mini (Small Language Model, deployed locally)
Inference Runtime	Ollama
Vision Model	Granite 3.2 Vision (via Ollama, for selective visual-page analysis)

Component	Configuration
RAG Framework	Retrieval-Augmented Generation (custom local pipeline)
Embedding Model	BGE-small-en-v1.5 (384-dimensional)
Vector Database	Redis 8.2.7 (Vector Sets — VADD / VSIM)
Programming Language	Python
Application Interface	Electron + React (desktop application, "FRIDAY")
Execution Environment	Local desktop environment, no cloud connection required

3.6 Research Variables

Independent Variables: embedding model; document chunk size; chunk overlap; Top-k retrieval value; similarity threshold; context size.

Dependent Variables: retrieval precision; retrieval recall; response latency; hallucination rate; knowledge-grounding quality; CPU utilization; memory consumption; response relevance.

3.7 Evaluation Metrics

Table 4: Performance Evaluation Metrics

Metric	Purpose
Precision	Measures the proportion of retrieved information that is relevant.
Recall	Measures the proportion of relevant information successfully retrieved.
Response Latency	Measures the time required to generate a response.
Hallucination Rate	Measures unsupported or factually inconsistent generated content.
Knowledge Grounding	Measures consistency between generated responses and retrieved evidence.
CPU Utilization	Measures computational load during local inference.
Memory Consumption	Measures RAM utilization during retrieval and generation.
Response Relevance	Measures the relevance of generated responses to user queries.
Privacy-Policy Success	Measures whether guardrail and privacy checks correctly block unsafe input/output.

3.8 Privacy Module and Threat Model

Because FRIDAY is intended to operate entirely on local hardware without a cloud connection, a lightweight privacy and guardrail module is integrated directly into the response pipeline. Its responsibilities include: (i) local-only processing, so that documents, queries, and generated responses never leave the device; (ii) input screening for prompt-injection attempts before a query reaches the

retrieval or generation stage; (iii) output screening ("Security Check") that verifies privacy protection, prompt-injection protection, and both input and output guardrails before a response is returned to the user; and (iv) an explicit "Local Mode" status indicator confirming that no cloud connection is required for the current session.

The privacy threat model considered during evaluation includes: unauthorized local access to the device or the vector store; prompt-injection attempts that try to override system instructions or reveal internal prompts; unintended retrieval of sensitive information for queries where it is not contextually relevant; and stale or revoked information remaining available after it should have been forgotten. The guardrail layer specifically addresses prompt-injection attempts by detecting patterns such as instructions to ignore prior system directives, blocking the query before it reaches the local SLM, and returning a safe, generic refusal rather than processing the request, as illustrated in Figure 7. A subsequent, unrelated query passes all four checks — privacy protection, prompt-injection protection, input guardrail, and output guardrail — as shown in Figure 8, confirming that the guardrail layer does not interfere with legitimate use.

Local execution is treated as reducing the need for cloud-based data transmission rather than as an absolute guarantee of device security; the evaluated threat model is limited to the guardrail behaviours implemented in the current prototype.

3.9 Reproducibility

The implementation documents the selected SLM (Phi-3 Mini), vision model (Granite 3.2 Vision), embedding model (BGE-small-en-v1.5), vector database (Redis 8.2.7), chunking configuration, Top-k values, and inference runtime (Ollama). Hardware specifications, software versions, and experimental parameters were recorded for each benchmark run, and the same evaluation query set was used across latency and Top-k experiments so that observed differences can be attributed to the investigated configuration rather than to changes in the evaluation environment. Wherever possible, open-source models and libraries were used to enable independent replication.

4. Results and Interpretation

The FRIDAY prototype was implemented as a privacy-preserving local Retrieval-Augmented Generation system. The implementation keeps document processing, embedding generation, vector retrieval, guardrail enforcement, and knowledge access entirely within the local computing environment. This section reports the measured results of the baseline implementation.

4.1 Knowledge Corpus and Document Processing

A local knowledge corpus of 12 PDF documents related to retrieval-augmented generation, local language models, desktop assistants, and voice-based intelligent systems was assembled and processed automatically, as shown in Figure 3. The processing pipeline extracted textual content, detected document images, and generated structured records for subsequent retrieval.

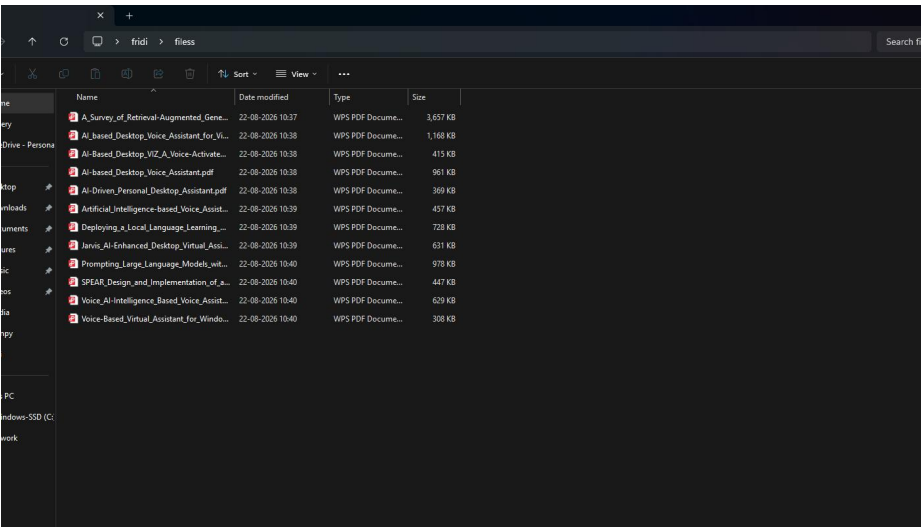


Figure 3: Local PDF knowledge corpus used to build the FRIDAY vector store (12 documents).

Table 5: FRIDAY Knowledge Corpus Statistics

Parameter	Value
PDF documents	12
Total pages	64
Detected images	63
Generated text chunks	843
Embedding model	BGE-small-en-v1.5
Embedding dimension	384
Vector database	Redis 8.2.7
Vector storage	Redis Vector Sets

4.2 Visual Document Processing

Because research documents frequently contain diagrams, tables, and architectural figures, the ingestion pipeline incorporated selective visual processing using a locally hosted Granite 3.2 Vision model through Ollama, applied only to pages containing meaningful visual information. This produced 27 visual-analysis records; a small number of requests experienced local inference timeouts, but the system continued processing subsequent documents rather than terminating the complete ingestion run. Figure 4 shows the ingestion process completing successfully across the full 843-chunk corpus.

```
PS C:\Users\siddh\Desktop\FRIDAY-RAG> python src/database/ingest_embeddings.py
Inserted: 736/843
Inserted: 752/843
Inserted: 768/843
Inserted: 784/843
Inserted: 800/843
Inserted: 816/843
Inserted: 832/843
Inserted: 843/843

=====
INGESTION COMPLETE
=====
Total chunks      : 843
Vectors inserted  : 843
Embedding time    : 30.94 sec
Total ingestion   : 31.48 sec
Throughput        : 26.78 chunks/sec
Vector Set        : friday_vectors
=====
```

Figure 4: FRIDAY embedding-ingestion pipeline completing for all 843 chunks.

4.3 Vector Embedding and Redis Storage

Each knowledge chunk was converted into a 384-dimensional, normalized, FP32 vector using BGE-small-en-v1.5 and stored locally using Redis 8.2.7 Vector Sets, whose VADD and VSIM commands provided vector insertion and similarity search. A total of 843 embeddings were successfully inserted into the `friday_vectors` vector set.

Table 6: Embedding and Vector Storage Performance

Metric	Measured Value
Chunks processed	843
Vectors inserted	843
Embedding dimension	384
Embedding time	30.94 s
Total ingestion time	31.48 s
Ingestion throughput	26.78 chunks/s
Vector set	friday_vectors

4.4 Baseline Semantic Retrieval

The baseline retrieval system converts the user query into a 384-dimensional BGE embedding and submits it to the Redis Vector Set using VSIM, with an initial Top-k of 5. For the query “What is Retrieval-Augmented Generation and how does it reduce hallucination?”, FRIDAY returned five highly similar chunks from the RAG survey document, with similarity scores of 0.8969, 0.8921, 0.8841, 0.8839, and 0.8839. Figure 5 shows the FRIDAY desktop application generating a grounded response to this query, citing its retrieved source and reporting a response time of 0.98 s using the local Phi-3 Mini model.



Figure 5: FRIDAY generating a locally grounded response with cited source and response time (Local Mode).

The results demonstrate that the vector-retrieval stage identifies semantically related content from the local corpus. However, some lower-ranked chunks contained references or broader conclusion material rather than direct explanatory evidence, motivating the Top-k optimization experiments in Section 4.6.

Table 7: Representative Baseline Retrieval Results

Rank	Chunk	Similarity
1	chunk_00007	0.896909
2	chunk_00014	0.892147
3	chunk_00047	0.884146
4	chunk_00082	0.883940
5	chunk_00066	0.883878

4.5 Retrieval Latency Evaluation

A ten-query benchmark measured the computational latency of the baseline retrieval pipeline, separately timing query-embedding generation and Redis vector retrieval. Figure 6 shows a representative benchmark run.

```
PS C:\Users\siddh\Desktop\FRIDAY-RAG> python src/optimization/retrieval_benchmark.py
Question: What methods can be used to evaluate the quality of RAG responses?
Embedding latency : 0.0185 sec
Retrieval latency : 0.000767 sec
Total latency      : 0.0193 sec
Retrieved chunks   : 5

Similarity scores:
1. chunk_00066 → 0.879427
2. chunk_00056 → 0.879417
3. chunk_00053 → 0.878566
4. chunk_00014 → 0.876361
5. chunk_00054 → 0.875103

=====
BENCHMARK COMPLETE
=====
Average embedding latency : 0.0203 sec
Average retrieval latency : 0.002930 sec
Average total latency     : 0.0232 sec
Top-k                     : 5
PS C:\Users\siddh\Desktop\FRIDAY-RAG>
```

Figure 6: FRIDAY retrieval-latency benchmark: per-query embedding/retrieval timing and similarity scores.

The benchmark produced an average embedding latency of 0.0203 s and an average Redis retrieval latency of 0.002930 s, giving an average retrieval-stage latency of approximately 0.0232 s per query. These measurements indicate that Redis vector search contributes only a small portion of total retrieval latency, while query-embedding generation is the larger component.

Table 8: Baseline Retrieval Latency

Metric	Average
Embedding latency	0.0203 s
Redis retrieval latency	0.002930 s
Total retrieval latency	0.0232 s
Top-k	5
Evaluation queries	10

4.6 Top-k Retrieval Optimization

The first retrieval-optimization experiment investigated the effect of the number of retrieved candidates, using $k = 3, 5, 8$, and 10. Increasing k increases the number of retrieved candidates but does not necessarily increase the semantic quality of every returned chunk: for the evaluated query, the highest similarity scores were concentrated among the first few candidates, while lower-ranked candidates showed progressively lower similarity. This is important for a local cognitive agent because retrieving unnecessary context increases downstream processing and can introduce irrelevant information into the generation stage.

Table 9: Top-k Retrieval Experiment

Top-k	Retrieval Time (s)	Results
3	0.013765	3
5	0.000839	5
8	0.000765	8
10	0.000697	10

The first $k = 3$ measurement showed a higher retrieval time than subsequent measurements, treated as a possible warm-up effect rather than evidence that a smaller k is inherently slower. The experiment establishes a baseline for further optimization rather than declaring a single optimal k from one query.

4.7 Privacy and Guardrail Validation

Because FRIDAY is designed as a privacy-first local agent, the guardrail layer described in Section 3.8 was validated with adversarial and benign queries. Figure 7 shows a query containing a prompt-injection attempt (an instruction to ignore prior system instructions and reveal the system prompt); the input was blocked by the security layer before reaching the local model, and the query was not processed.



Figure 7: FRIDAY detecting and blocking a prompt-injection attempt at the input-guardrail stage.

Figure 8 shows a subsequent security check confirming that privacy protection, prompt-injection protection, the input guardrail, and the output guardrail all passed for a legitimate query, with a reported response time of 3.20 s, indicating that guardrail evaluation does not prevent normal use of the system.



Figure 8: FRIDAY security-check panel confirming all four guardrail checks passed.

4.8 Baseline System Summary

The completed baseline implementation successfully established a fully local retrieval pipeline for the FRIDAY cognitive-agent prototype. The system processed 12 PDF documents and generated 843 knowledge chunks, converted into 384-dimensional embeddings using BGE-small-en-v1.5 and stored in Redis 8.2.7 Vector Sets. The measured ingestion throughput was 26.78 chunks per second. During the ten-query retrieval benchmark, average embedding latency was 20.3 ms, while Redis vector retrieval required approximately 2.93 ms on average. Guardrail checks for prompt-injection and privacy protection were confirmed to operate correctly on both adversarial and benign queries. These measurements establish a quantitative baseline for further optimization.

4.9 Discussion

The implementation demonstrates the feasibility of constructing a local RAG pipeline for a privacy-preserving cognitive agent using small and locally executable components. Using a local document corpus avoids uploading knowledge documents to an external cloud-based retrieval service; the embedding stage converts documents into compact vector representations, while Redis Vector Sets provide a local mechanism for semantic similarity search.

The experimental results indicate that vector retrieval itself is computationally lightweight compared with query embedding: the average Redis retrieval latency (~2.93 ms) was substantially lower than the average embedding latency (~20.3 ms). Future optimization should therefore also investigate the embedding stage and retrieval quality, not only the vector database. The Top-k experiment further revealed that increasing the number of retrieved candidates does not guarantee that every additional candidate is useful, since lower-ranked chunks may contain references or broadly related information rather than direct evidence for the query. This motivates the next research stage, in which similarity thresholding, improved chunking, hybrid retrieval, lightweight reranking, and context reduction can be evaluated systematically against the baseline established here.

The guardrail validation additionally shows that privacy-preserving checks can be layered onto a local RAG pipeline without disabling normal functionality, supporting H3. The current implementation should therefore be considered a measured, privacy-aware baseline rather than the final optimized system; each future optimization can be evaluated against this baseline using retrieval quality, latency, and context-efficiency metrics.

5. Conclusion

This research proposed and implemented an optimization-oriented methodology for Retrieval-Augmented Generation in privacy-preserving local cognitive agents using Small Language Models. Rather than treating RAG as a fixed pipeline, the methodology treats retrieval, embedding generation, document chunking, vector indexing, Top-k selection, context construction, privacy guardrails, and local inference as optimization variables, evaluated through an applied, quantitative, and experimental research design.

The FRIDAY prototype successfully established the foundation of a privacy-preserving local RAG pipeline. A corpus of 12 PDF documents (64 pages) was processed into 843 knowledge chunks, embedded using BGE-small-en-v1.5, and stored as 384-dimensional vectors in Redis 8.2.7 Vector Sets; the complete embedding and ingestion process required 31.48 s at a throughput of 26.78 chunks per second. The ten-query retrieval benchmark produced an average embedding latency of 20.3 ms and an average Redis retrieval latency of 2.93 ms, for an average retrieval-stage latency of approximately 23.2 ms. The initial Top-k experiments confirmed that the system can efficiently retrieve semantically related information while revealing the need for relevance filtering and context optimization, and the guardrail validation confirmed that prompt-injection detection and privacy/security checks operate correctly without disrupting legitimate use.

These results provide the experimental baseline for the next stage of FRIDAY research, in which retrieval quality and computational efficiency will be jointly optimized — through similarity thresholding, hybrid retrieval, reranking, and refined chunking — while maintaining fully local, privacy-preserving operation. The expected contribution is a systematic and reproducible methodology for optimizing RAG pipelines specifically for resource-constrained Small Language Models, providing a foundation for efficient, knowledge-grounded, and privacy-preserving local cognitive agents.

Acknowledgement

The author would like to express sincere gratitude to Sanjivani University, Department of Artificial Intelligence and Data Science, for providing the academic environment and resources required to conduct this research. The author also acknowledges the researchers whose prior work on retrieval-augmented generation, semantic embeddings, on-device language models, and privacy-preserving intelligent agents contributed to the foundation of this study.

References

- [1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-T. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020.
- [2] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, Y. Sun, M. Wang, and H. Wang, “Retrieval-Augmented Generation for Large Language Models: A Survey,” *arXiv preprint arXiv:2312.10997*, 2023.
- [3] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in *Proc. 2019 Conf. on Empirical Methods in Natural Language Processing and the 9th Int. Joint Conf. on Natural Language Processing*, pp. 3982–3992, 2019.
- [4] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the Middle: How Language Models Use Long Contexts,” *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.
- [5] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, and Z. Ling, “On-Device Language Models: A Comprehensive Review,” *arXiv preprint arXiv:2409.00088*, 2024.
- [6] C. Lee, D. Prahlad, D. Kim, and H. Kim, “Work-in-Progress: On-device Retrieval Augmented Generation with Knowledge Graphs for Personalized Large Language Models,” in *Proc. 24th ACM SIGBED Int. Conf. on Embedded Software (EMSOFT)*, 2024, doi: 10.1109/EMSOFT60242.2024.00006.
- [7] Z. Yu, S. Liu, P. Denny, A. Bergen, and M. Liut, “Integrating Small Language Models with Retrieval-Augmented Generation in Computing Education: Key Takeaways, Setup, and Practical Insights,” in *Proc. 56th ACM Technical Symp. on Computer Science Education*, pp. 1302–1308, 2025, doi: 10.1145/3641554.3701844.
- [8] S. Zeng, J. Zhang, P. He, J. Ren, T. Zheng, H. Lu, H. Xu, H. Liu, Y. Xing, and J. Tang, “Mitigating the Privacy Issues in Retrieval-Augmented Generation (RAG) via Pure Synthetic Data,” in *Proc. 2025 Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 24527–24558, 2025, doi: 10.18653/v1/2025.emnlp-main.1247.
- [9] D. Ru, L. Qiu, X. Hu, T. Zhang, P. Shi, S. Chang, C. Jiayang, C. Wang, S. Sun, H. Li, Z. Zhang, B. Wang, J. Jiang, T. He, Z. Wang, P. Liu, Y. Zhang, and Z. Zhang, “RAGChecker: A Fine-grained Framework for Diagnosing Retrieval-Augmented Generation,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024, doi: 10.52202/079017-0692.
- [10] T. Jung and I. Joe, “An Intelligent Docent System with a Small Large Language Model (sLLM) Based on Retrieval-Augmented Generation (RAG),” *Applied Sciences*, vol. 15, no. 17, Art. no. 9398, 2025, doi: 10.3390/app15179398.